Sur les notions de classes, d'objets, d'héritages, de surcharge des méthodes, de polymorphisme, voir – en plus du support de cours – les références sur le web, notamment :

- Introduction à la programmation objet (https://hdd34.developpez.com/cours/artpoo/) ;
- POO à gogo – Les méthodes avec Free Pascal / Lazarus (1/2) (https://gilles-vasseur.developpez.com/tutoriels/poo-les-methodes-1/)

1. Gestion d'un véhicule – Classe et objet

Ecrire une application qui permet à un utilisateur de saisir les caractéristiques d'un véhicule, puis de les afficher par la suite en incluant la taxe calculée à partir de ses propriétés.

Dans le module « **UVoiture.pas** », définir la classe **TVoiture**. Voici sa description :

```
TVoiture = class
  //champs
  marque : shortstring;
  age : integer;
  cv : integer;
  //méthodes
  procedure saisie();//saisie des paramètres
  procedure affichage();//affichage des paramètres
  function taxe(): double;//taxe du véhicule
end;
```

Les méthodes répondent au cahier des charges suivant :

- saisie() permet de saisir les propriétés de l'objet (marque, âge, cv [chevaux fiscaux]) ;
- affichage() affiche ces propriétés ainsi que la taxe associée au véhicule ;
- taxe() retourne une valeur calculée comme suit, taxe = 30.0 * cv + 5.0 * âge

Dans le programme principal, vous devez tout à tour :

- Initialiser un objet de type TVoiture ;
- Faire appel à saisie() ;
- Puis à affichage() ;
- Détruire l'objet.

Voici un exemple d'exécution :

```
Saisie d'une voiture :
Marque = Renault
Age = 5
```

```
Puissance fiscale = 7

Affichage d'une voiture :
Marque = Renault
Age = 5
Puissance fiscale = 7
Taxe =  2.3500000000000000E+002
```

2. Gérer différents types de voitures – Héritage

Nous souhaitons pouvoir gérer deux types de véhicules : les voitures « normales » telles que définies dans l'exercice précédent ; les « utilitaires » qui incluent une caractéristique supplémentaire (le cubage) et dont la taxe est calculée différemment.

Voici la description de la classe **TUtilitaire**, qui doit être héritière de TVoiture (n'allez pas faire des copier-coller intempestifs !) et définie dans un nouveau module « **UVoitureUtilitaire.pas** » :

```
TUtilitaire = class(TVoiture)
  //champ supplémentaire - cubage
  cubage: integer;
  //surcharge de saisie, affichage et taxe
  procedure saisie();//saisie des paramètres
  procedure affichage();//affichage des paramètres
  function taxe(): double;//taxe du véhicule
end;
```

Remarque :

- Saisie() et Affichage() doivent intégrer le champ « cubage » maintenant.
- La taxe est obtenue en additionnant « 0.01 * cubage » à la taxe usuelle.

Dans le programme principal, vous demandez à l'utilisateur le type de voiture qu'il souhaite instancier. Vous faites saisir alors ses paramètres (avec « cubage » si un utilitaire est demandé). Vous en affichez ensuite ses caractéristiques en incluant la taxe calculée correctement.

Voici un exemple d'exécution :

```
Type de vehicule (1 : utilitaire, autre : normale) : 1

Saisie d'une voiture :
Marque = Opel
Age = 6
Puissance fiscale = 10
Cubage : 1000

Affichage d'une voiture :
Marque = Opel
Age = 6
```

```
Puissance fiscale = 10
Taxe =  3.3000000000000000E+002
Cubage : 1000
```

3. Polymorphisme

Dans le programme principal de l'exercice précédent, vous avez dû jongler avec 2 variables de types différents pour pouvoir appréhender les différents types de véhicules que l'utilisateur pouvait potentiellement choisir. Modifiez les déclarations des classes (**TVoiture** et **TUtilitaire**) pour que vous puissiez rédiger une nouvelle version du programme principal – largement simplifié par rapport à celui de l'exercice précédent – comme suit :

```
var v: TVoiture; //une seule variable
    code: integer;//type de véhicule

begin
  //demander le type de véhicule
  write('Type de vehicule (1 : utilitaire, autre : normale) : '); readln(code);
  //initialisation selon le type de véhicule
  if (code <> 1)
   then v:= TVoiture.create()
   else v:= TUtilitaire.create();
  //appel des méthodes
  v.saisie();
  v.affichage();
  v.free();
  //blocage console
  readln;
end.
```

N.B. : On remarquera ici qu'un seul objet « v », de type « TVoiture », est censé pouvoir représenter les deux classes lors de l'appel du constructeur. Les méthodes saisie() et affichage() appelées doivent être en rapport avec la classe réellement initialisée.

4. Collection polymorphe

On souhaite compléter notre application de manière à gérer une collection de véhicules qui peuvent être de type « TVoiture » ou « Tutilitaire ». Dans un nouveau module « **UCollecVoiture.pas** », définissez et implémenter la classe **TCollecVoitures** qui se présente comme suit :

```
TYPE
```

```
  //tableau dynamique de voitures
  TTabVoiture = array of TVoiture;

  //classe de gestion de collection de voitures
  { TCollecVoitures }
  TCollecVoitures = class
    //champ interne collection de tableau de voitures
    liste: TTabVoiture;
    //méthodes
    constructor create();//constructeur
    destructor destroy(); override;//destructeur
    procedure ajout(v: TVoiture);//ajouter une voiture dans la collection
    procedure affichage();//affichage de la collection
    function SommeTaxes(): double;//somme des taxes des véhicules
  end;
```

Remarque :

- Notez l'utilisation du champ interne « liste » de type TTabVoiture, un tableau dynamique de véhicules TVoiture ou de toute autre classe héritière.
- Le constructeur create() et le destructor destroy() doivent se charger respectivement de l'initialisation et de la destruction de la liste (et des véhicules qu'elle contient).
- La procédure ajout() doit permettre de rajouter une voiture déjà instanciée dans la liste, une TVoiture ou une TUtilitaire.

Dans le programme principal, tant que l'utilisateur indique souhaiter ajouter une voiture dans la collection (c.-à-d. il ne répond pas « STOP » à la question d'ajouter une nouvelle voiture) : vous demandez le type de véhicule à instancier (TVoiture ou TUtilitaire), vous en faites saisir les caractéristiques, et vous l'ajoutez à la collection. Lorsqu'il indique « STOP », le programme doit afficher la liste des véhicules avec leurs caractéristiques, il doit aussi afficher la somme de leurs taxes.

Voici un exemple d'exécution :

```
Saisie nouvelle voiture (STOP ou autre) ?
Type du vehicule (1 : utilitaire, autre : normal) ? 0
Saisie d'une voiture :
Marque = Opel
Age = 5
Puissance fiscale = 12

Saisie nouvelle voiture (STOP ou autre) ?
Type du vehicule (1 : utilitaire, autre : normal) ? 1
Saisie d'une voiture :
Marque = Iveco
```

```
Age = 3
Puissance fiscale = 15
Cubage : 1500

Saisie nouvelle voiture (STOP ou autre) ? STOP

>>> Affichage de la liste des vehicules <<<
**
Affichage d'une voiture :
Marque = Opel
Age = 5
Puissance fiscale = 12
Taxe =  3.8500000000000000E+002
**
Affichage d'une voiture :
Marque = Iveco
Age = 3
Puissance fiscale = 15
Taxe =  4.8000000000000000E+002
Cubage : 1500

Somme des taxes =  8.6500000000000000E+002
```